# **Documento de Diseño y Arquitectura: Aplicación Modular sobre USB-NCM**

Versión: 1.9 (Final Merge)

Fecha: 2025-08-05

Plataforma: ESP-IDF v5.5

Objetivo: Comunicación TCP/IP sobre USB nativo (ESP32-S3) mediante clase USB-NCM/ECM, con servicios modulares (MQTT, HTTP futuro), orquestados por app\_manager\_aq y desacoplados del medio físico mediante esp\_netif.

## **1. Resumen Ejecutivo**

Este documento define una arquitectura modular para habilitar red IP sobre USB en ESP32-S3 usando TinyUSB (clase NCM preferida; ECM y RNDIS como *fallbacks* opcionales). La lógica de aplicación reside en app\_manager\_aq, que arranca y detiene servicios especializados (p. ej. mqtt\_service\_aq) y gestiona cambios de interfaz (*USB ↔ Wi-Fi*) de forma explícita.

**Cambios clave v1.9 (Merge)**

* **Sincronización de API TinyUSB:** Se sustituye tud\_network\_link\_state\_cb por tud\_mount\_cb/tud\_umount\_cb + tud\_suspend\_cb/tud\_resume\_cb (relevante para TinyUSB ≥ v0.18.0).
* **Gestión de Eventos:** Se formaliza un bus de eventos USB\_NET\_EVENTS y una política de *reconnect* coordinada desde app\_manager\_aq.
* **Estrategia MQTT Robusta:** Se adopta la estrategia de bind() del socket a la IP del esp\_netif de USB para forzar la interfaz de red, alineada con las prácticas recomendadas de ESP-IDF.
* **Optimización de Rendimiento:** Se incluyen recomendaciones específicas para el *tuning* de lwIP, colas de transmisión (TX) y copiado de búferes de recepción (RX) a pbuf.
* **Pruebas y CI:** Se define una matriz de pruebas multi-OS y una estrategia de Integración Continua para IDF 5.4/5.5.

**Compatibilidad OS**

* **Windows 10/11:** NCM funciona de forma nativa. **Windows <10** puede requerir un archivo .inf manual.
* **macOS y Linux:** NCM/ECM funcionan de forma nativa. ECM puede ser preferible para compatibilidad con versiones antiguas de macOS.
* **Rendimiento Esperado:** El puerto USB del ESP32-S3 es **Full Speed (12 Mbps)**. El *throughput* real típico en NCM es de **1–6 Mbps**, dependiendo del MTU, coalescing, nivel de logging y el tuning de lwIP.

## **2. Arquitectura del Sistema**

### **2.1 Modelo de capas y flujo**

+-----------------+  
| app\_main.c | Punto de entrada mínimo  
+-----------------+  
 |  
 v  
+-----------------------------+  
| app\_manager\_aq | Orquestación y lógica de app  
+-----------------------------+  
 | |  
 | (Inicia y | (Inyecta dependencia  
 | obtiene netif) | y notifica eventos)  
 v v  
+-----------+ +-----------------+  
| usb\_comms | | mqtt\_service\_aq | Servicios especializados  
| \_aq | +-----------------+  
+-----------+

* **Capa inferior (app\_main)**: inicialización mínima (NVS, esp\_event\_loop\_create\_default()) y delegación a app\_manager\_aq.
* **Capa intermedia (servicios)**: componentes *caja negra* que encapsulan una funcionalidad (USB-NCM, MQTT).
* **Capa superior (app\_manager\_aq)**: orquesta el arranque/parada ordenada, maneja errores y gestiona el cambio de interfaz de red (*netif switching*).

### **2.2 Concurrencia y escalabilidad**

* **usb\_comms\_aq**: debe correr en una tarea dedicada con **prioridad 5** y un **stack ≥ 4096 bytes** para procesar datos de red sin latencia.
* **mqtt\_service\_aq**: puede correr en una tarea separada con menor prioridad (ej. 3).
* **Pinning (Opcional)**: para aplicaciones sensibles al jitter, se puede fijar la tarea de usb\_comms\_aq a un núcleo específico (xTaskCreatePinnedToCore), pero se debe medir el impacto primero.
* **Múltiples netif**: El app\_manager\_aq debe poder gestionar un cambio de interfaz (ej. app\_manager\_switch\_netif(esp\_netif\_t\*)). Este cambio debe publicar un evento NETIF\_SWITCHED, y los servicios *stateful* (como MQTT) **deben cerrar sus sockets y reconectar** al recibirlo.

## **3. Diseño detallado de componentes**

### **3.1 app\_manager\_aq**

**Responsabilidad:** orquestar el ciclo de vida de la aplicación, la política de netif activa y la propagación de eventos y errores.

**API pública (include/app\_manager\_aq.h)**

#pragma once  
#include "esp\_netif.h"  
void app\_manager\_start(void);  
void app\_manager\_stop(void);  
void app\_manager\_switch\_netif(esp\_netif\_t\* new\_netif);

### **3.2 usb\_comms\_aq — Conectividad USB (NCM/ECM/RNDIS)**

**Responsabilidad:** abstraer la complejidad del hardware USB y exponer un esp\_netif estándar y funcional.

**API pública (include/usb\_comms\_aq.h)**

#pragma once  
#include "esp\_netif.h"  
esp\_err\_t usb\_comms\_start(void);  
esp\_netif\_t\* usb\_comms\_get\_netif\_handle(void);

Manejo de eventos USB (TinyUSB ≥ v0.18.0)

Debe usar tud\_mount\_cb(), tud\_umount\_cb(), tud\_suspend\_cb() y tud\_resume\_cb() para detectar cambios de estado y emitir eventos USB\_NET\_\* hacia app\_manager\_aq.

ESP\_EVENT\_DECLARE\_BASE(USB\_NET\_EVENTS);  
typedef enum { USB\_NET\_UP, USB\_NET\_DOWN, USB\_NET\_SUSPEND, USB\_NET\_RESUME } usb\_net\_event\_id\_t;  
  
void tud\_mount\_cb(void) { esp\_event\_post(USB\_NET\_EVENTS, USB\_NET\_UP, NULL, 0, 0); }  
void tud\_umount\_cb(void) { esp\_event\_post(USB\_NET\_EVENTS, USB\_NET\_DOWN, NULL, 0, 0); }  
// etc.

**Rendimiento (TX/RX)**

* **TX (NCM):** usar tinyusb\_net\_send\_async() con una **cola** (xQueueCreate) para gestionar el flujo (*back-pressure*), condicionando el envío a tud\_ready().
* **RX:** **copiar siempre** los datos entrantes a un búfer pbuf de LWIP antes de llamar a esp\_netif\_receive() para desacoplar los búferes de la pila USB.

### **3.3 mqtt\_service\_aq**

**Responsabilidad:** cliente MQTT agnóstico al medio, con control explícito de la interfaz de red.

**API pública (include/mqtt\_service\_aq.h)**

#pragma once  
#include "esp\_netif.h"  
  
// ... (definiciones de callback y struct de configuración)  
  
esp\_err\_t mqtt\_service\_start(const mqtt\_service\_config\_t \*config);  
// ... (otras funciones)

Selección de interfaz (enlace a esp\_netif)

Dado que esp-mqtt no tiene una API pública estable para fijar el netif, la estrategia recomendada es vincular el socket a la IP local del esp\_netif deseado antes de conectar.

// Dentro del transporte personalizado o antes de conectar  
int sock = /\* socket del transporte \*/;  
int ifx = esp\_netif\_get\_netif\_impl\_index(config->netif\_handle);  
setsockopt(sock, IPPROTO\_IP, IP\_BOUND\_IF, &ifx, sizeof(ifx));  
// Opcionalmente, bind() explícito a la IP del netif  
connect(sock, ...);

## **4. Flujo de datos**

### **4.1 Diagrama de secuencia (TX asíncrona)**

app\_manager\_aq -> mqtt\_service\_aq: publish("topic","data")  
mqtt\_service\_aq -> lwIP: send()  
lwIP -> esp\_netif: .transmit(packet)  
esp\_netif -> usb\_comms\_aq: netif\_transmit(packet)  
usb\_comms\_aq -> TinyUSB: tinyusb\_net\_send\_async(packet)  
TinyUSB -> Host PC: [USB Data]

### **4.2 Manejo de fallos y optimización**

* **Unplug durante TX**: El evento USB\_NET\_DOWN debe provocar que se baje el esp\_netif, se notifique al app\_manager\_aq y se prepare una reconexión futura.
* **Coalescing**: Investigar la agrupación de paquetes en NCM para reducir la sobrecarga del USB.

## **5. Gestión de eventos, errores y logging**

* **Bucle de eventos**: Declarar bases de eventos (USB\_NET\_EVENTS, APP\_EVENTS) y registrar manejadores con esp\_event\_handler\_instance\_register() para poder desregistrarlos de forma segura.
* **Errores**: Propagar esp\_err\_t y permitir que app\_manager\_aq implemente políticas de reintento con *exponential backoff*.
* **Logging**: Usar ESP\_LOGx con TAG único. Evitar ESP\_LOG\_BUFFER\_HEX en caminos de código críticos para el rendimiento.

## **6. Pruebas y robustez**

* **Unit tests (Unity)**: Usar mocks de esp\_netif\_receive/transmit y simular los callbacks de TinyUSB (tud\_mount\_cb, etc.).
* **Integración**: Probar con iperf3 y Wireshark en Windows, macOS y Linux. Los casos de prueba deben incluir desconexiones, suspensión del host y ráfagas de datos.
* **CI/CD**: Usar GitHub Actions con una matriz para IDF {5.4, 5.5} y versiones fijadas de esp\_tinyusb.

## **7. Configuración del proyecto**

**idf\_component.yml (ejemplo)**

dependencies:  
 idf: ">=5.4.0,<6.0.0"  
 espressif/esp\_tinyusb: ">=1.7.0,<2.0.0"  
 espressif/esp-mqtt: "~1.5"

**Kconfig (extracto)**

menu "USB Network Configuration"  
 config USB\_NET\_CLASS  
 string "USB Network Class"  
 default "NCM"  
 help  
 Options: NCM, ECM, RNDIS  
  
 config USB\_NET\_IPV4\_ADDR  
 string "Static IPv4 Address"  
 default "192.168.7.1"  
endmenu

## **8. Consideraciones futuras**

* **Gestión de energía**: En tud\_suspend\_cb, reducir la frecuencia del reloj para entrar en modos de bajo consumo.
* **Fallback de red**: Implementar la lógica en app\_manager\_aq para priorizar USB y cambiar a Wi-Fi si USB\_NET\_DOWN es detectado.
* **Seguridad**: Usar mTLS para MQTT/HTTP, con certificados cargados de forma segura desde NVS cifrada.

## **9. Referencias**

* ESP-IDF v5.5: *Get Started* y *Changelog*.
* esp\_tinyusb: Guía de clases ECM/NCM/RNDIS y ejemplos.
* TinyUSB: *Release notes* v0.18.0 (reescritura NCM y cambios en callbacks).
* Ejemplo usb-netif con DHCP y logs.
* Discusiones sobre *socket binding* a esp\_netif (IP\_BOUND\_IF, esp\_netif\_get\_netif\_impl\_index).

## **10. Apéndice: Parámetros y Notas Operativas**

* **Rangos IP**: Evitar 192.168.137.0/24 (ICS de Windows).
* **Tuning lwIP**: Aumentar PBUF\_POOL\_SIZE si se observan paquetes perdidos (*drops*).
* **RNDIS**: Usar solo por compatibilidad heredada; requiere un control de flujo (*back-pressure*) estricto en la transmisión.
* **Métricas**: Implementar contadores para *drops*, *retries*, latencia de TX y profundidad de la cola de transmisión para monitorear el rendimiento.